

On Cascade Products of Answer Set Programs

CHRISTIAN ANTIC^{*}

*Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
(e-mail: christian.antic@icloud.com)*

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Describing complex objects by elementary ones is a common strategy in mathematics and science in general. In their seminal 1965 paper, Kenneth Krohn and John Rhodes showed that every finite deterministic automaton can be represented (or “emulated”) by a cascade product of very simple automata. This led to an elegant algebraic theory of automata based on finite semigroups (Krohn-Rhodes Theory). Surprisingly, by relating logic programs and automata, we can show in this paper that the Krohn-Rhodes Theory is applicable in Answer Set Programming (ASP). More precisely, we recast the concept of a cascade product to ASP, and prove that every program can be represented by a product of very simple programs, the reset and standard programs. Roughly, this implies that the reset and standard programs are the basic building blocks of ASP with respect to the cascade product. In a broader sense, this paper is a first step towards an algebraic theory of products and networks of nonmonotonic reasoning systems based on Krohn-Rhodes Theory, aiming at important open issues in ASP and AI in general.

KEYWORDS: Answer Set Programming (ASP), Krohn-Rhodes Theory, Automata

1 Introduction

Describing complex objects by elementary ones is a common strategy in mathematics and science in general. For instance, the fundamental theorem of number theory states that every natural number can be (uniquely) represented by its prime factors. Similarly, in their seminal 1965 paper “Algebraic theory of machines, I. Prime decomposition theorem for finite semigroups and machines”, Kenneth Krohn and John Rhodes showed that every finite deterministic automaton can be represented (or “emulated”) by a cascade product of very simple automata. This led to an elegant algebraic theory of automata based on finite semigroups (Krohn-Rhodes Theory) and, more recently, to an algebraic theory of networks of automata (cf. Dömösi and Nehaniv (2005)).

Answer Set Programming (ASP) (Gelfond and Lifschitz 1991), on the other hand, has become a prominent knowledge representation and reasoning (KR&R) formalism over the last two decades, with a wide range of applications in AI-related subfields such as, e.g., nonmonotonic reasoning, diagnosis, and planning (cf. Brewka et al. (2011)).

In this paper, we aim at combining these two vivid areas of research and will show that, surprisingly, the Krohn-Rhodes Theory *is* applicable in ASP. More precisely, we

^{*} This work was supported by the Austrian Science Fund (FWF) under grant S11409-N23.

recast the concept of a cascade product to ASP, and prove that every program can be represented by a product of reset programs $R = \{1 \leftarrow \text{not } 1\}$ and n -standard programs S_n consisting only of rules of the simple form $i \leftarrow j$, not k (cf. Theorem 4.3). Roughly, this implies that the reset and standard programs are the basic building blocks of ASP with respect to the cascade product and, strikingly, while the reset and standard programs do not possess any interesting declarative meaning (the reset program is inconsistent and the standard programs have only the empty answer set), their interaction can “emulate” any given program. In other words, the product semantics *emerges* from the interplay of its (simple) factors and allows for arbitrary complex behavior.

To the best of our knowledge, this is the first paper applying the Krohn-Rhodes Theory to logic programming. In a broader sense, it is a first step towards an algebraic theory of products and networks of nonmonotonic reasoning systems based on Krohn-Rhodes Theory, with far-reaching potential application areas including some important open issues in ASP and AI in general (cf. the discussion in Section 6).

The rest of the paper is structured as follows. In Section 2, we present the basic definitions and results concerning ASP and automata. In Section 3, we introduce the concept of a programmable automaton, and show that the distinguished reset and standard automata are programmable in this sense. In Section 4, the main part of this paper, we recast the concept of a cascade product to ASP and prove that every program can be (homomorphically) represented by reset and standard programs. In Section 5, we study the more restricted type of isomorphic representation and provide a complete class of programs with respect to it; moreover, we show that positive tight programs are isomorphically representable by reset programs. Finally, in Section 6, we conclude with a discussion on interesting lines for future research.

2 Preliminaries

We assume that the reader is familiar with the concept of a partially ordered set and that of a (complete) lattice. Following (Gécseg 1986), we denote by $[n]$, $n \geq 0$, the set $\{1, \dots, n\}$. We denote, for $k \geq 1$ and $i \geq 0$, the least residue of i modulo n by $i \bmod n$. For a set X , we denote by $|X|$ the cardinality of X . Given a function $f : X \times Y \rightarrow Z$, we denote by $f(\cdot, y)$ the function from X into Z mapping each $x \in X$ to $f(x, y) \in Z$, and we denote by $\text{lfp } f(\cdot, y)$ the least fixpoint of $f(\cdot, y)$. We denote the power set of X by $\mathfrak{P}(X)$.

2.1 Answer Set Programs

We briefly recall the syntax and answer set semantics (Gelfond and Lifschitz 1991) of nonmonotonic logic programs in an operator-based setting (cf. Denecker et al. (2000)).

Syntax In the sequel, Γ will denote a finite nonempty set of propositional atoms. A (*normal logic*) *program* P over some Γ_P is a finite nonempty set of rules of the form

$$a \leftarrow b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m, \quad m \geq k \geq 0, \quad (1)$$

where $a, b_1, \dots, b_m \in \Gamma_P$ and *not* denotes *negation-as-failure*. For convenience, we define for a rule r of the form (1), $H(r) = a$, $B^+(r) = \{b_1, \dots, b_k\}$, $B^-(r) = \{b_{k+1}, \dots, b_m\}$,

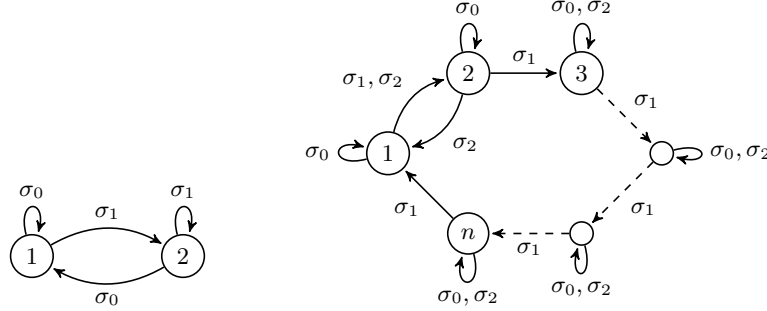


Fig. 1. The (two-state) reset automaton \mathfrak{R} and the n -standard automaton \mathfrak{S}_n .

and $B(r) = B^+(r) \cup B^-(r)$. We call r a *fact*, if $B(r) = \emptyset$; and we call r *positive* if $B^-(r) = \emptyset$. We say that P is *positive* if every rule $r \in P$ is positive, and we call P *tight* if there is a mapping ℓ from Γ_P into the nonnegative integers such that for each rule r in P , $\ell(H(r)) > \ell(b)$ for every $b \in B^+(r)$.

Semantics An *interpretation* of P is any subset $I \subseteq \Gamma_P$ and we denote the set of all interpretations of P by $\mathcal{I}_P = \mathfrak{P}(\Gamma_P)$. Define the \mathcal{I} -valued immediate consequence operator $\Psi_P : \mathcal{I}_P \times \mathcal{I}_P \rightarrow \mathcal{I}_P$ by

$$\Psi_P(I, J) = \{H(r) : r \in P, B^+(r) \subseteq I, B^-(r) \cap J = \emptyset\}.$$

Intuitively, $\Psi_P(I, J)$ contains the heads $H(r)$ of all rules r in P where the positive part of the body evaluates to *true* in I , and the negative part evaluates to *true* in J . Given some $I \in \mathcal{I}_P$, it is well-known that $\Psi_P(\cdot, I)$ is monotone on the complete lattice \mathcal{I}_P ordered by \subseteq , and hence has a *least fixpoint* denoted by $\text{lfp } \Psi_P(\cdot, I)$. We say that $I \in \mathcal{I}_P$ is an *answer set* of P , or a Ψ_P -*answer set*, if $I = \text{lfp } \Psi_P(\cdot, I)$.

2.2 Krohn-Rhodes Theory

In this section, we recall some basic definitions and results of Krohn-Rhodes Theory by mainly following the lines of (Gécseg 1986, Chapters 1–3).

An *automaton* $\mathfrak{A} = \langle Q, \Sigma, \delta \rangle$ consists of a finite set Q of *states*, a finite nonempty set Σ , called the *input alphabet*, and a mapping $\delta : Q \times \Sigma \rightarrow Q$ called the *transition function*.

Given two automata $\mathfrak{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathfrak{A}' = \langle Q', \Sigma', \delta' \rangle$, we say that \mathfrak{A}' is a *subautomaton* of \mathfrak{A} if $Q' \subseteq Q$, $\Sigma' \subseteq \Sigma$, and δ' is the restriction of δ to $Q' \times \Sigma'$. A pair $h = (h_1, h_2)$ of surjective mappings $h_1 : Q \rightarrow Q'$, $h_2 : \Sigma \rightarrow \Sigma'$ is a *homomorphism* of \mathfrak{A} onto \mathfrak{A}' if $h_1(\delta(q, x)) = \delta'(h_1(q), h_2(x))$, for every $q \in Q, x \in \Sigma$. The pair h is an *isomorphism* if h_1 and h_2 are bijective homomorphisms, and we say that \mathfrak{A} is *isomorphic* to \mathfrak{A}' if there exists an isomorphism h of \mathfrak{A} onto \mathfrak{A}' . If $\Sigma = \Sigma'$, then we omit h_2 and define $h = h_1$.

An equivalence relation \sim on Q is a *congruence relation* of \mathfrak{A} if $q \sim q'$ implies $\delta(q, x) \sim \delta(q', x)$, for all $q, q' \in Q$ and $x \in \Sigma$. We denote the congruence class of $q \in Q$ with respect to \sim by q/\sim , and define the *quotient automaton* $\mathfrak{A}/\sim = \langle Q/\sim, \Sigma, \delta/\sim \rangle$ by $\delta/\sim(q/\sim, x) = \delta(q, x)/\sim$ for all $q \in Q$ and $x \in \Sigma$. Conversely, given a homomorphism $h = (h_1, h_2)$ of \mathfrak{A}

onto \mathfrak{A}' , we mean by the *congruence relation of \mathfrak{A} induced by h* the binary relation \sim on Q given by $q \sim q'$ if $h_1(q) = h_1(q')$.

The following automata will play a central role throughout the rest of the paper (cf. Figure 1):

1. Define the (*two-state*) *reset automaton* $\mathfrak{R} = \langle [2], \{\sigma_0, \sigma_1\}, \delta_{\mathfrak{R}} \rangle$ by $\delta_{\mathfrak{R}}(i, \sigma_0) = 1$, and $\delta_{\mathfrak{R}}(i, \sigma_1) = 2$, for all $i \in [2]$.
2. We call an automaton $\mathfrak{S} = \langle [n], \{\sigma_0, \sigma_1, \sigma_2\}, \delta_{\mathfrak{S}} \rangle$, $n > 1$, *standard* if $\delta_{\mathfrak{S}}$ satisfies the following conditions, for all $i \in [n]$:
 - (a) $\delta_{\mathfrak{S}}(i, \sigma_0) = i$;
 - (b) $\delta_{\mathfrak{S}}(i, \sigma_1) = (i \bmod n) + 1$;
 - (c) $\delta_{\mathfrak{S}}(i, \sigma_2) = \begin{cases} 2 & \text{if } i = 1, \\ 1 & \text{if } i = 2, \\ i & \text{otherwise.} \end{cases}$

We denote the n -state standard automaton by \mathfrak{S}_n .

The following operators on arbitrary classes \mathcal{A} of automata will be useful:

1. $\mathbf{S}(\mathcal{A})$ denotes the set of subautomata of automata from \mathcal{A} ;
2. $\mathbf{H}(\mathcal{A})$ denotes the homomorphic images of automata from \mathcal{A} ;
3. $\mathbf{I}(\mathcal{A})$ denotes the isomorphic images of automata from \mathcal{A} .

We will write $\mathbf{XY}(\mathcal{A})$ for $\mathbf{X}(\mathbf{Y}(\mathcal{A}))$, where \mathbf{X} and \mathbf{Y} are operators from above.

We now define the cascade product for automata, which is also known as the wreath (Krohn and Rhodes 1965) or α_0 -product (Gécseg 1986) in the literature.

Definition 2.1 (Cascade Automata Product) For some $k > 0$, let $\mathfrak{A}_i = \langle Q_i, \Sigma_i, \delta_i \rangle$, $i \in [k]$, be a family of automata, and let Σ be an alphabet. A *feedforward function* for $\mathfrak{A}_1, \dots, \mathfrak{A}_k$ is a mapping $\psi : (Q_1 \times \dots \times Q_k) \times \Sigma \rightarrow \Sigma_1 \times \dots \times \Sigma_k$ with

$$\psi((q_1, \dots, q_k), \sigma) = (\psi_1((q_1, \dots, q_k), \sigma), \dots, \psi_k((q_1, \dots, q_k), \sigma))$$

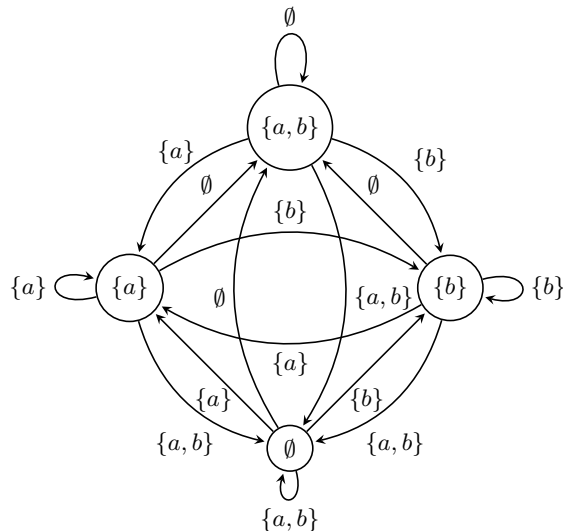
where the *component feedforward function* ψ_i , $i \in [k]$, is a mapping from $(Q_1 \times \dots \times Q_k) \times \Sigma$ into Σ_i . In the sequel, we omit those arguments q_j , $j \in [k]$, ψ_i does not depend on. The *cascade* (or *loop-free*) *automata product* of $\mathfrak{A}_1, \dots, \mathfrak{A}_k$ with respect to $\Sigma_{\mathfrak{A}} = \Sigma$ and some feedforward function $\psi_{\mathfrak{A}}$

$$\mathfrak{A} = \langle Q_{\mathfrak{A}}, \Sigma_{\mathfrak{A}}, \delta_{\mathfrak{A}} \rangle = \mathfrak{A}_1 \times \dots \times \mathfrak{A}_k [\Sigma_{\mathfrak{A}}, \psi_{\mathfrak{A}}]$$

is given by $Q_{\mathfrak{A}} = Q_1 \times \dots \times Q_k$ where ψ_i , $i \in [k]$, is independent of its j^{th} component, $j \in [k]$, whenever $j \geq i$. Finally, we define the *transition function* $\delta_{\mathfrak{A}} : Q_{\mathfrak{A}} \times \Sigma_{\mathfrak{A}} \rightarrow Q_{\mathfrak{A}}$ by

$$\delta_{\mathfrak{A}}((q_1, \dots, q_k), \sigma) = (\delta_1(q_1, \psi_1(\sigma)), \dots, \delta_k(q_k, \psi_k((q_1, \dots, q_{k-1}), \sigma))).$$

Definition 2.2 We say that an automaton \mathfrak{A} *homomorphically* (resp., *isomorphically*) *represents* an automaton \mathfrak{A}' if $\mathfrak{A}' \in \mathbf{HS}(\{\mathfrak{A}\})$ (resp., $\mathfrak{A}' \in \mathbf{IS}(\{\mathfrak{A}\})$). Moreover, we say that a class \mathcal{A} of automata is *homomorphically* (resp., *isomorphically*) *complete* with respect to the cascade automata product if every automaton \mathfrak{A} can be homomorphically (resp., isomorphically) represented by a cascade automata product of automata from \mathcal{A} .



The following result is a consequence of the Krohn-Rhodes decomposition theorem (Krohn and Rhodes 1965), and it will be of great importance for our main Theorem 4.3.

We now turn to isomorphic completeness. Let $\mathfrak{T}_n = \langle [n], \Sigma_n, \delta_n \rangle$, $n \geq 1$, such that Σ_n is the set of all mappings $\sigma : [n] \rightarrow [n]$, and $\delta_n(j, \sigma) = \sigma(j)$, for all $j \in [n]$.

3 Programmable Automata

Given some program P , we define its *characteristic automaton* $\mathfrak{A}_P = \langle Q_P, \Sigma_P, \delta_P \rangle$ by $Q_P = \Sigma_P = \mathcal{I}_P$ and $\delta_P = \Psi_P$. In the sequel, we will not distinguish between the operator Ψ_P and the characteristic automaton $\mathfrak{A}_P = \langle \mathcal{I}_P, \mathcal{I}_P, \Psi_P \rangle$, i.e., we will refer to \mathfrak{A}_P simply by Ψ_P and will call Ψ_P the characteristic automaton of P (cf. Figure 2).

Definition 3.1 We say that an automaton \mathfrak{A} is *homomorphically* (resp., *isomorphically*) *programmable* if there exists some program P such that Ψ_P homomorphically (resp., isomorphically) represents \mathfrak{A} , that is, $\mathfrak{A} \in \mathbf{HS}(\{\Psi_P\})$ (resp., $\mathfrak{A} \in \mathbf{IS}(\{\Psi_P\})$). We then say that P *homomorphically* (resp., *isomorphically*) *programs* \mathfrak{A} .

We illustrate this concept with an example; in Theorem 3.4 we will see that the reset automaton \mathfrak{R} and the n -state standard automaton \mathfrak{S}_n , $n > 1$, are isomorphically programmable.

Example 3.2 Define the *elevator automaton* $\mathfrak{E} = \langle [2], \{\sigma_0, \sigma_1\}, \delta_{\mathfrak{E}} \rangle$ by $\delta_{\mathfrak{E}}(1, \sigma_0) = 1$, $\delta_{\mathfrak{E}}(1, \sigma_1) = 2$, and $\delta_{\mathfrak{E}}(2, \sigma_0) = \delta_{\mathfrak{E}}(2, \sigma_1) = 2$ (cf. Dömösi and Nehaniv (2005, p.45)). On the other hand, define the *elevator program* E by $E = \{e \leftarrow e; e \leftarrow \text{not } e\}$. Then, $h = (h_1, h_2)$ defined by $h_1(\emptyset) = 1$, $h_1(\{e\}) = 2$, $h_2(\{e\}) = \sigma_0$, and $h_2(\emptyset) = \sigma_1$ is an isomorphism of (the automaton) Ψ_E onto \mathfrak{E} ; hence, E isomorphically programs \mathfrak{E} . \square

For convenience, in the sequel we occasionally denote atoms by nonnegative integers.

Definition 3.3 The *reset program* R over $\Gamma_R = [1]$ consists of the following single rule:

$$1 \leftarrow \text{not } 1.$$

The *n-standard program* (or *n-program*) S_n over $\Gamma_n = [n] \cup \{3\}$, $n > 1$, consists of the following rules, for all $i \in [n]$ and $j \in [n]$, $j > 2$:

$$\begin{array}{ll} i \leftarrow i, \text{ not } 1, & 1 \leftarrow 2, \text{ not } 3, \\ (i \bmod n) + 1 \leftarrow i, \text{ not } 2, & 2 \leftarrow 1, \text{ not } 3, \\ & j \leftarrow j, \text{ not } 3. \end{array}$$

Note that the reset program R is *inconsistent*, i.e., has no Ψ_R -answer sets, and for every $n > 1$, the n -program S_n has the Ψ_{S_n} -answer set \emptyset .

Theorem 3.4 *The reset program R and the n -standard program S_n isomorphically program the reset automaton \mathfrak{R} and the n -state standard automaton \mathfrak{S}_n , $n > 1$, respectively.*

Proof

Define $h_{R,1} : \mathcal{I}_R \rightarrow [2]$ and $h_{R,2} : \mathcal{I}_R \rightarrow \{\sigma_0, \sigma_1\}$ by $h_{R,1}(\emptyset) = 1$, $h_{R,1}(\{1\}) = 2$, $h_{R,2}(\emptyset) = \sigma_1$, and $h_{R,2}(\{1\}) = \sigma_0$. A straightforward computation shows that $h_R = (h_{R,1}, h_{R,2})$ is an isomorphism of Ψ_R onto \mathfrak{R} ; i.e., we have

$$h_{R,1}(\Psi_R(I, J)) = \delta_{\mathfrak{R}}(h_{R,1}(I), h_{R,2}(J)), \quad \text{for all } I, J \in \mathcal{I}_R.$$

Hence, $\mathfrak{R} \in \mathbf{IS}(\{\Psi_R\})$.

For the second part, let $\Psi'_{S_n} = \langle \mathcal{I}'_{S_n}, \mathcal{I}''_{S_n}, \Psi'_{S_n} \rangle$ be the subautomaton of Ψ_{S_n} given by $\mathcal{I}'_{S_n} = \{\{i\} : i \in [n]\} \subseteq \mathcal{I}_{S_n}$, $\mathcal{I}''_{S_n} = \{\{2, 3\}, \{1, 3\}, \{1, 2\}\} \subseteq \mathcal{I}_{S_n}$, and Ψ'_{S_n} equals Ψ_{S_n} restricted to $\mathcal{I}'_{S_n} \times \mathcal{I}''_{S_n}$. Define $h_{S_n,1} : \mathcal{I}'_{S_n} \rightarrow [n]$ by $h_{S_n,1}(\{i\}) = i$, for all $i \in [n]$; and $h_{S_n,2} : \mathcal{I}''_{S_n} \rightarrow \{\sigma_0, \sigma_1, \sigma_2\}$ by $h_{S_n,2}(\{2, 3\}) = \sigma_0$, $h_{S_n,2}(\{1, 3\}) = \sigma_1$, and $h_{S_n,2}(\{1, 2\}) = \sigma_2$. Then, $h = (h_{S_n,1}, h_{S_n,2})$ is an isomorphism of Ψ'_{S_n} onto \mathfrak{S}_n ; i.e., we have

$$h_{S_n,1}(\Psi'_{S_n}(\{i\}, J)) = \delta_{\mathfrak{S}_n}(h_{S_n,1}(\{i\}), h_{S_n,2}(J)), \quad \text{for all } i \in [n] \text{ and } J \in \mathcal{I}''_{S_n}.$$

Hence, $\mathfrak{S}_n \in \mathbf{IS}(\{\Psi_{S_n}\})$. \square

4 Cascade Products and Homomorphic Representations

In this section, we recast the concept of a cascade automata product presented in Section 2.2 (cf. Definition 2.1) to the setting of ASP and study homomorphic representations.

Definition 4.1 (Cascade Program Product) Let P_1, \dots, P_k , $k > 1$, be a family of programs over some alphabets $\Gamma_{P_1}, \dots, \Gamma_{P_k}$, respectively, and let $\mathcal{I}_{\mathbf{P}}$ be some finite nonempty set. A *feedforward function* for P_1, \dots, P_k is a mapping $\psi_{\mathbf{P}} : (\mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}) \times \mathcal{I}_{\mathbf{P}} \rightarrow \mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}$ with

$$\psi_{\mathbf{P}}((I_1, \dots, I_k), \mathbf{J}) = (\psi_{\mathbf{P},1}((I_1, \dots, I_k), \mathbf{J}), \dots, \psi_{\mathbf{P},k}((I_1, \dots, I_k), \mathbf{J}))$$

where the *component feedforward function* $\psi_{\mathbf{P},i}$, $i \in [k]$, is a mapping from $(\mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}) \times \mathcal{I}_{\mathbf{P}}$ into \mathcal{I}_{P_i} . In the sequel, we omit those arguments I_j , $j \in [k]$, $\psi_{\mathbf{P},i}$ does not depend on. The (*cascade* or *loop-free program*) *product* of P_1, \dots, P_k with respect to $\mathcal{I}_{\mathbf{P}}$ and some feedforward function $\psi_{\mathbf{P}}$

$$\mathbf{P} = P_1 \ltimes \dots \ltimes P_k [\mathcal{I}_{\mathbf{P}}, \psi_{\mathbf{P}}]$$

is given by its component feedforward functions $\psi_{\mathbf{P},i}$, $i \in [k]$, which are independent of their j^{th} component, $j \in [k]$, whenever $j \geq i$. Finally, we define the *characteristic automaton* $\Psi_{\mathbf{P}} = \langle Q_{\mathbf{P}}, \Sigma_{\mathbf{P}}, \Psi_{\mathbf{P}} \rangle$ of \mathbf{P} by $Q_{\mathbf{P}} = \mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}$, $\Sigma_{\mathbf{P}} = \mathcal{I}_{\mathbf{P}}$, and $\Psi_{\mathbf{P}} : (\mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}) \times \mathcal{I}_{\mathbf{P}} \rightarrow \mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}$ with

$$\Psi_{\mathbf{P}}((I_1, \dots, I_k), \mathbf{J}) = (\Psi_{P_1}(I_1, \psi_{\mathbf{P},1}(\mathbf{J})), \dots, \Psi_{P_k}(I_k, \psi_{\mathbf{P},k}((I_1, \dots, I_{k-1}), \mathbf{J}))).$$

Intuitively, a cascade program product is a collection of programs which are connected to each other and exchange (local) information via a feedforward function, where each component program may depend only on the preceding components and on the global input; every state-transition of the characteristic automaton of the product is then the result of the *simultaneous local* state-transitions of the characteristic automata of its component programs.

Formally, a product is not a program according to the definition given in Section 2.1. However, we can relate products and programs as follows (cf. Definition 2.2).

Definition 4.2 We say that a cascade program product \mathbf{P} *homomorphically* (resp., *isomorphically*) *represents* a program P if $\Psi_{\mathbf{P}}$ homomorphically (resp., isomorphically) represents Ψ_P , that is, $\Psi_P \in \mathbf{HS}(\{\Psi_{\mathbf{P}}\})$ (resp., $\Psi_P \in \mathbf{IS}(\{\Psi_{\mathbf{P}}\})$). Moreover, we say that a class \mathcal{P} of programs is *homomorphically* (resp., *isomorphically*) *complete* with respect to the cascade program product if every program P can be homomorphically (resp., isomorphically) represented by a cascade program product of programs from \mathcal{P} .

We now make the relation between products and programs more explicit. In the context of logic programming, representation (or “emulation”) means semantic equivalence (modulo some encoding). According to Definition 4.2, a product $\mathbf{P} = P_1 \ltimes \dots \ltimes P_k [\mathcal{I}_{\mathbf{P}}, \psi_{\mathbf{P}}]$, $k > 1$, represents a program P if the characteristic automaton $\Psi_{\mathbf{P}}$ represents the characteristic automaton Ψ_P (in the sense of Section 2.2); that is, if there exists a subautomaton $\Psi'_{\mathbf{P}} = \langle \mathcal{I}'_{P_1} \times \dots \times \mathcal{I}'_{P_k}, \mathcal{I}'_{\mathbf{P}}, \Psi'_{\mathbf{P}} \rangle$ of $\Psi_{\mathbf{P}}$ and a congruence relation \sim on $\mathcal{I}'_{P_1} \times \dots \times \mathcal{I}'_{P_k}$ such that $\Psi'_{\mathbf{P}}/\sim$ is isomorphic to Ψ_P . Intuitively, every interpretation $I \in \mathcal{I}_{\mathbf{P}}$ of P then corresponds to a congruence class of k -tuples from $\mathcal{I}'_{P_1} \times \dots \times \mathcal{I}'_{P_k}$; if the representation is isomorphic, then I can be identified with a single k -tuple (I'_1, \dots, I'_k) and in this case we can imagine (I'_1, \dots, I'_k) to be an “encoding” of I .

Interestingly enough, by the forthcoming Theorem 4.3, we can assume that only reset and standard programs occur as factors in the product \mathbf{P} . That is, Theorem 4.3 roughly implies that by knowing the reset program R and all the n -programs S_n , $n > 1$, and by

knowing how to form the cascade program product, we essentially know all programs; viz., the reset and standard programs are the basic building blocks of ASP with respect to the cascade program product.

We are now ready to state the main theorem of this paper.

Theorem 4.3 *Every program P over some alphabet Γ_P , with $|\Gamma_P| = m$, can be homomorphically represented by a cascade program product \mathbf{P} of reset and 2^m -standard programs.*

Proof

According to Definition 4.2, we have to show that there exists some product \mathbf{P} such that $\Psi_{\mathbf{P}}$ homomorphically represents Ψ_P . Since Ψ_P has 2^m states, Theorem 2.3 yields a cascade automata product $\mathfrak{A}_P = \mathfrak{A}_1 \ltimes \dots \ltimes \mathfrak{A}_k [\mathcal{I}_P, \psi_P]$, for some $k > 0$, consisting of reset and 2^m -standard automata homomorphically representing Ψ_P . Note that \mathfrak{A}_P has the same input alphabet \mathcal{I}_P as Ψ_P . Define the product $\mathbf{P} = P_1 \ltimes \dots \ltimes P_k [\mathcal{I}_{\mathbf{P}}, \psi_{\mathbf{P}}]$ as follows: (i) for every $i \in [k]$, if \mathfrak{A}_i is the reset automaton \mathfrak{R} (resp., 2^m -standard automaton \mathfrak{S}_{2^m}), then P_i is the reset program R (resp., 2^m -standard program S_{2^m}); (ii) $\mathcal{I}_{\mathbf{P}}$ is the input alphabet \mathcal{I}_P of \mathfrak{A}_P and Ψ_P ; (iii) $\psi_{\mathbf{P}}$ is a mapping from $(\mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}) \times \mathcal{I}_P$ into $\mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}$ where \mathcal{I}_{P_i} , $i \in [k]$, is \mathcal{I}_R (resp., \mathcal{I}_{2^m}) if P_i is the reset program R (resp., 2^m -standard program S_{2^m}), and $\psi_{\mathbf{P},i}$ coincides with $\psi_{P,i}$ on the appropriate subset of $\mathcal{I}_{P_1} \times \dots \times \mathcal{I}_{P_k}$ modulo the isomorphisms defined in the proof of Theorem 3.4. Then, it follows from Theorem 3.4 that $\Psi_{\mathbf{P}}$ isomorphically represents \mathfrak{A}_P and, by transitivity of representation, it homomorphically represents Ψ_P , which proves our theorem. \square

It is worth noting that the proof of Theorem 4.3 yields a product \mathbf{P} whose characteristic automaton $\Psi_{\mathbf{P}}$ has the same input alphabet \mathcal{I}_P as the characteristic automaton Ψ_P of P . Therefore, we can characterize the answer sets of P by $\Psi_{\mathbf{P}}$ as follows. Roughly, the product semantics of \mathbf{P} emerges as an interaction of its (simple) factors P_1, \dots, P_k with respect to P . More precisely, by the remarks given above, there exists a quotient subautomaton $\Psi'_{\mathbf{P}}/\sim$ of $\Psi_{\mathbf{P}}$ which is isomorphic to Ψ_P and which has the same input alphabet as Ψ_P . Let $h : \mathcal{I}'_{P_1} \times \dots \times \mathcal{I}'_{P_k} \rightarrow \mathcal{I}_P$ be the corresponding homomorphism of $\Psi'_{\mathbf{P}}$ onto Ψ_P inducing \sim ; we order $(\mathcal{I}'_{P_1} \times \dots \times \mathcal{I}'_{P_k})/\sim$ by $(I_1, \dots, I_k)/\sim \subseteq_h (I'_1, \dots, I'_k)/\sim$ if $h(I_1, \dots, I_k) \subseteq h(I'_1, \dots, I'_k)$. Then, $\langle (\mathcal{I}'_{P_1} \times \dots \times \mathcal{I}'_{P_k})/\sim, \subseteq_h \rangle$ is isomorphic (as a lattice) to $\langle \mathcal{I}_P, \subseteq \rangle$, and we say that $I \in \mathcal{I}_P$ is a $\Psi'_{\mathbf{P}}/\sim$ -answer set if $I = h(\text{lfp } \Psi'_{\mathbf{P}}/\sim(\cdot, I))$. Then, we have the following correspondence:

$$I \text{ is a } \Psi_P\text{-answer set} \Leftrightarrow I \text{ is a } \Psi'_{\mathbf{P}}/\sim\text{-answer set.} \quad (2)$$

By Theorem 4.3, we can assume that in the right hand side of (2), only reset and 2^m -standard programs occur.

We illustrate these concepts by giving some examples.

Example 4.4 Let $A = \{a \leftarrow\}$ be a program consisting of a single fact. We can interpret A as a database *storing* some information represented by a . Observe that neither the reset program R nor the 2-program S_2 contains a *fact*. However, we verify that

$$\mathbf{A} = R[\mathcal{I}_A, \psi_{\mathbf{A}}] = \{1 \leftarrow \text{not } 1\}[\mathcal{I}_A, \psi_{\mathbf{A}}]$$

defined by $\psi_{\mathbf{A}}(J) = \emptyset$, for all $J \in \mathcal{I}_A$, isomorphically represents A . Define $h : \mathcal{I}_R \rightarrow \mathcal{I}_A$

by $h(\emptyset) = \emptyset$ and $h(\{1\}) = \{a\}$. We check that h is an isomorphism:

$$\begin{aligned} h(\Psi_{\mathbf{A}}(I, J)) &= h(\Psi_R(I, \psi_{\mathbf{A}}(J))) \\ &= h(\Psi_R(I, \emptyset)) \\ &= h(\{1\}) \\ &= \{a\} \\ &= \Psi_A(h(I), J) \end{aligned}$$

holds for all $I \in \mathcal{I}_R$ and $J \in \mathcal{I}_A$. Therefore, the congruence relation \sim induced by h is the trivial diagonal relation and $\Psi_{\mathbf{A}}/\sim$ is isomorphic to $\Psi_{\mathbf{A}}$. Hence, $\Psi_A \in \mathbf{IS}(\{\Psi_{\mathbf{A}}\})$. The calculation above proves that $\{a\}$ is the only Ψ_A -answer set or, equivalently, the only $\Psi_{\mathbf{A}}$ -answer set. Intuitively, \mathbf{A} “emulates” the storage of the fact a by ignoring the input J appropriately. Generally, the program $A_m = \{a_1 \leftarrow \dots; a_m \leftarrow\}$, $m \geq 1$, is isomorphically represented by $\mathbf{A}_m = R \ltimes \dots \ltimes R[\mathcal{I}_{A_m}, \psi_{\mathbf{A}_m}]$ (with m factors) where $\psi_{\mathbf{A}_m, i}((I_1, \dots, I_{i-1}), J) = \emptyset$, for all $i \in [m]$, $I_1, \dots, I_{i-1} \in \mathcal{I}_R$, and $J \in \mathcal{I}_{A_m}$. Here, an isomorphism is an arbitrary “binary encoding” h of \mathcal{I}_{A_m} ; e.g., $h(I_1, \dots, I_m) = \{a_i \in \mathcal{I}_{A_m} : I_i = \{1\}, i \in [m]\}$. \square

Example 4.5 The program $B = \{a \leftarrow \text{not } b; b \leftarrow \text{not } a\}$ (cf. Figure 2) is isomorphically represented by the cascade program product

$$\mathbf{B} = R \ltimes R[\mathcal{I}_B, \psi_{\mathbf{B}}] = \{1 \leftarrow \text{not } 1\} \ltimes \{1 \leftarrow \text{not } 1\}[\mathcal{I}_B, \psi_{\mathbf{B}}]$$

defined by

$$\begin{aligned} \psi_{\mathbf{B},1}(\emptyset) &= \psi_{\mathbf{B},1}(\{a\}) = \emptyset, & \psi_{\mathbf{B},2}(I, \emptyset) &= \psi_{\mathbf{B},2}(I, \{b\}) = \emptyset, \\ \psi_{\mathbf{B},1}(\{b\}) &= \psi_{\mathbf{B},1}(\{a, b\}) = \{1\}, & \psi_{\mathbf{B},2}(I, \{a\}) &= \psi_{\mathbf{B},2}(I, \{a, b\}) = \{1\}, \end{aligned}$$

for all $I \in \mathcal{I}_R$. Let $h : \mathcal{I}_R \times \mathcal{I}_R \rightarrow \mathcal{I}_B$ be the “binary encoding” of \mathcal{I}_B given by $h(\emptyset, \emptyset) = \emptyset$, $h(\{1\}, \emptyset) = \{a\}$, $h(\emptyset, \{1\}) = \{b\}$, and $h(\{1\}, \{1\}) = \{a, b\}$. It is straightforward to verify that h is an isomorphism of $\Psi_{\mathbf{B}}$ onto Ψ_B . For instance, we compute:

$$\begin{aligned} h(\Psi_{\mathbf{B}}((\emptyset, \emptyset), \{a\})) &= h(\Psi_R(\emptyset, \psi_{\mathbf{B},1}(\{a\})), \Psi_R(\emptyset, \psi_{\mathbf{B},2}(\emptyset, \{a\}))) \\ &= h(\Psi_R(\emptyset, \emptyset), \Psi_R(\emptyset, \{1\})) \\ &= h(\{1\}, \emptyset) \\ &= \{a\} \\ &= \Psi_B(h(\emptyset, \emptyset), \{a\}). \end{aligned}$$

Hence, $\Psi_B \in \mathbf{IS}(\{\Psi_{\mathbf{B}}\})$. By the remarks given above, I is a Ψ_B -answer set iff I is a $\Psi_{\mathbf{B}}$ -answer set and, clearly, $\{a\}$ and $\{b\}$ are the only ones. \square

5 Isomorphic Representations

In this section, we study the more restricted type of isomorphic representation and provide a complete class of programs with respect to it. Moreover, in Theorem 5.3 we show that every positive tight program can be isomorphically represented by a cascade program product of reset programs.

For some $n \geq 1$, let $\sigma_1, \dots, \sigma_{n^n}$ be an enumeration of the set of all mappings from

$[n]$ into $[n]$. Define T_n over $\Gamma_{T_n} = [n^n]$ to be the program consisting of the rules, for all $j \in [n]$ and $k \in [n^n]$:

$$\sigma_k(j) \leftarrow j, \text{ not } k.$$

As a consequence of Theorem 2.4, we obtain the following completeness result.

Theorem 5.1 *The class of programs consisting of all T_n , $n \geq 1$, is isomorphically complete with respect to the cascade program product.*

Proof

According to Theorem 2.4 and Definition 4.2, we have to show that for every $n \geq 1$, the automaton $\mathfrak{T}_n = \langle [n], \Sigma_n, \delta_n \rangle$ can be embedded into a cascade automata product of Ψ_{T_n} with a single factor. Define $\Psi_{\mathbf{T}_n} = \Psi_{T_n} [\mathcal{I}_{T_n}, \psi_{\mathbf{T}_n}]$ by $\psi_{\mathbf{T}_n}(J) = J$, for all $J \in \mathcal{I}_{T_n}$. Define the embedding $h = (h_1, h_2)$, with $h_1 : [n] \rightarrow \mathcal{I}_{T_n}$ and $h_2 : \Sigma_n \rightarrow \mathcal{I}_{T_n}$, by $h_1(j) = \{j\}$ and $h_2(\sigma_k) = \{1, \dots, k-1, k+1, \dots, n^n\}$, $k \in [n^n]$. Clearly, h_1 and h_2 are one-one, and the following computation proves that h is indeed an embedding:

$$\begin{aligned} \Psi_{\mathbf{T}_n}(h_1(j), h_2(\sigma_k)) &= \Psi_{T_n}(h_1(j), \psi_{\mathbf{T}_n}(h_2(\sigma_k))) \\ &= \Psi_{T_n}(h_1(j), h_2(\sigma_k)) \\ &= \Psi_{T_n}(\{j\}, \{1, \dots, k-1, k+1, \dots, n^n\}) \\ &= \{\sigma_k(j)\} \\ &= h_1(\sigma_k(j)) \\ &= h_1(\delta_n(j, \sigma_k)) \end{aligned}$$

holds for all $j \in [n]$ and $k \in [n^n]$. \square

We now turn to the restricted class of positive (i.e., negation-free) tight programs.

Example 5.2 Consider the positive tight program $C = \{a \leftarrow; b \leftarrow a; c \leftarrow a, b\}$. The product $\mathbf{C} = R \ltimes R \ltimes R [\mathcal{I}_C, \psi_{\mathbf{C}}]$ given by

$$\psi_{\mathbf{C},1}(J) = \emptyset \quad \psi_{\mathbf{C},2}(I_1, J) = \{1\} - I_1 \quad \psi_{\mathbf{C},3}((I_1, I_2), J) = \{1\} - (I_1 \cap I_2)$$

for all $I_1, I_2 \in \mathcal{I}_R$ and $J \in \mathcal{I}_C$, isomorphically represents C . Again, we define the isomorphism h to be a “binary encoding” of \mathcal{I}_C where, e.g., $(\{1\}, \emptyset, \emptyset)$ is mapped to $\{a\}$, $(\{1\}, \emptyset, \{1\})$ is mapped to $\{a, c\}$ and so on. For instance, we can compute the least model $I = \{a, b, c\}$ of C as follows:

$$\begin{aligned} h(\Psi_{\mathbf{C}}((\emptyset, \emptyset, \emptyset), J)) &= h(\Psi_R(\emptyset, \emptyset), \Psi_R(\emptyset, \{1\}), \Psi_R(\emptyset, \{1\})) = h(\{1\}, \emptyset, \emptyset) = \{a\} \\ h(\Psi_{\mathbf{C}}((\{1\}, \emptyset, \emptyset), J)) &= h(\Psi_R(\{1\}, \emptyset), \Psi_R(\emptyset, \emptyset), \Psi_R(\emptyset, \{1\})) = h(\{1\}, \{1\}, \emptyset) = \{a, b\} \\ h(\Psi_{\mathbf{C}}((\{1\}, \{1\}, \emptyset), J)) &= h(\Psi_R(\{1\}, \emptyset), \Psi_R(\{1\}, \emptyset), \Psi_R(\emptyset, \emptyset)) = h(\{1\}, \{1\}, \{1\}) = I \\ h(\Psi_{\mathbf{C}}((\{1\}, \{1\}, \{1\}), J)) &= h(\Psi_R(\{1\}, \emptyset), \Psi_R(\{1\}, \emptyset), \Psi_R(\{1\}, \emptyset)) = h(\{1\}, \{1\}, \{1\}) = I \end{aligned}$$

where $J \in \mathcal{I}_C$ is arbitrary. The calculation shows that I is a $\Psi_{\mathbf{C}}$ -answer set or, equivalently, a $\Psi_{\mathbf{C}}$ -answer set and, clearly, it is the only one.

Now consider the slightly different program $C' = \{a \leftarrow; b \leftarrow a; c \leftarrow a; c \leftarrow b\}$. Then, C' is isomorphically represented by the product $\mathbf{C}' = R \ltimes R \ltimes R [\mathcal{I}_{C'}, \psi_{\mathbf{C}'}]$ given by

$$\psi_{\mathbf{C}',1}(J) = \emptyset \quad \psi_{\mathbf{C}',2}(I_1, J) = \{1\} - I_1 \quad \psi_{\mathbf{C}',3}((I_1, I_2), J) = \{1\} - (I_1 \cup I_2)$$

for all $I_1, I_2 \in \mathcal{I}_R$ and $J \in \mathcal{I}_{C'}$. Let h be defined as before. Iterating $\Psi_{C'}$ bottom-up as above yields, for all $J \in \mathcal{I}_{C'}$:

$$\begin{aligned} h(\Psi_{C'}((\emptyset, \emptyset, \emptyset), J)) &= h(\Psi_R(\emptyset, \emptyset), \Psi_R(\emptyset, \{1\}), \Psi_R(\emptyset, \{1\})) = h(\{1\}, \emptyset, \emptyset) = \{a\} \\ h(\Psi_{C'}((\{1\}, \emptyset, \emptyset), J)) &= h(\Psi_R(\{1\}, \emptyset), \Psi_R(\emptyset, \emptyset), \Psi_R(\emptyset, \emptyset)) = h(\{1\}, \{1\}, \{1\}) = I \\ h(\Psi_{C'}((\{1\}, \{1\}, \{1\}), J)) &= h(\Psi_R(\{1\}, \emptyset), \Psi_R(\{1\}, \emptyset), \Psi_R(\{1\}, \emptyset)) = h(\{1\}, \{1\}, \{1\}) = I \end{aligned}$$

which shows that I is also a $\Psi_{C'}$ -answer set or, equivalently, a $\Psi_{C'}$ -answer set. \square

It is straightforward to generalize Example 5.2 to the general case.

Theorem 5.3 *Every positive tight program P can be isomorphically represented by a cascade program product of reset programs.*

6 Discussion and Conclusion

In this paper, we applied the Krohn-Rhodes Theory (Krohn and Rhodes 1965), presented here following (Gécseg 1986), to Answer Set Programming (ASP) (Gelfond and Lifschitz 1991). Particularly, we defined a cascade product for ASP and, by relating programs and automata, showed that every program can be represented (or “emulated”) by a product of very simple programs. We thus obtained nice theoretical results regarding the structure of ASP programs, which can be straightforwardly generalized to wider classes of non-monotonic reasoning formalisms. More precisely, as our concepts and results hinge on the operator Ψ_P , they can be directly reformulated in the algebraic framework of Approximation Fixpoint Theory (AFT) (Denecker et al. 2000), which captures, e.g., ordinary ASP, default and autoepistemic logic (Denecker et al. 2003), and ASP with external sources (Antić et al. 2013).

In a broader sense, this paper is a first step towards an algebraic theory of products and networks of nonmonotonic reasoning systems, including ASP and other formalisms. More precisely, we considered here only the very restricted (though powerful) kind of *cascade* product; it corresponds to the α_0 -product in (Gécseg 1986), and to the wreath product in finite semigroup theory (Krohn and Rhodes 1965). In the automata literature, however, many other important products have been studied (for an overview see Dömösi and Nehaniv (2005)). We believe that recasting these kinds of products to ASP will lead to interesting results. Particularly, the notion of an asynchronous network (cf. Dömösi and Nehaniv (2005, Chapter 7)) seems very appealing from an ASP point of view, as current modular ASP formalisms (e.g., Dao-Tran et al. (2009)) cannot cope with asynchronous module structures according to our knowledge. Moreover, as different formalisms can be unified in the AFT-setting, heterogeneous networks in the vein of multi-context systems (cf. Brewka et al. (2011)) arise naturally. Finally, our concept of a product semantics *emerging* from the interaction of its simple factors (cf. Section 4) seems interesting from a general AI perspective and we believe that it deserves a more intensive (and probably more intuitive) study in future work.

Although the Krohn-Rhodes decomposition theorem (Krohn and Rhodes 1965) is now almost 50 years old, implementations and feasible applications of the Krohn-Rhodes Theory emerged only very recently (cf. Egri-Nagy and Nehaniv (2005)); our paper provides further evidence that it is a valuable tool for knowledge representation and reasoning

in AI (e.g., Egri-Nagy and Nehaniv (2006)), and implementations in the ASP-setting remain as future work.

Acknowledgements I am grateful to Michael Fink and Hans Tompits for valuable suggestions which improved the presentation of the paper. I thank the anonymous referees for useful comments.

References

- ANTIĆ, C., EITER, T., AND FINK, M. 2013. HEX semantics via approximation fixpoint theory. In *LPNMR 2013*, P. Cabalar and S. T. Cao, Eds. LNAI 8148. Springer-Verlag, Berlin/Heidelberg, 102–115.
- BREWKA, G., EITER, T., AND FINK, M. 2011. Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, M. Balduccini and T. C. Son, Eds. LNAI 6565. Springer-Verlag, Berlin/Heidelberg, 233–258.
- BREWKA, G., EITER, T., AND TRUSZCZYŃSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.
- DAO-TRAN, M., EITER, T., FINK, M., AND KRENNWALLNER, T. 2009. Modular nonmonotonic logic programming revisited. In *ICLP 2009*. LNCS 5649. Springer-Verlag, Berlin, 145–159.
- DENECKER, M., MAREK, V. W., AND TRUSZCZYŃSKI, M. 2000. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In *Logic-based Artificial Intelligence*, J. Minker, Ed. Kluwer Academic Publishers, Dordrecht, 127–144.
- DENECKER, M., MAREK, V. W., AND TRUSZCZYŃSKI, M. 2003. Uniform semantic treatment of default and autoepistemic logic. *Artificial Intelligence* 143, 1, 79–122.
- DÖMÖSI, P. AND NEHANIV, C. L. 2005. *Algebraic Theory of Automata Networks: An Introduction*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA.
- EGRI-NAGY, A. AND NEHANIV, C. L. 2005. Algebraic hierarchical decomposition of finite state automata: Comparison of implementations for Krohn-Rhodes theory. In *CIAA 2004*, M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, Eds. LNCS 3317. Springer-Verlag, Berlin/Heidelberg, 315–316.
- EGRI-NAGY, A. AND NEHANIV, C. L. 2006. Making sense of the sensory data - coordinate systems by hierarchical decomposition. In *KES 2006, Part III*, B. Gabrys, R. J. Howlett, and L. C. Jain, Eds. LNAI 4253. Springer-Verlag, Berlin/Heidelberg, 330–340.
- GÉCSEG, F. 1986. *Products of Automata*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin/Heidelberg.
- GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3-4, 365–385.
- KROHN, K. AND RHODES, J. 1965. Algebraic theory of machines, I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society* 116, 450–464.